**Name of the game: Chess**

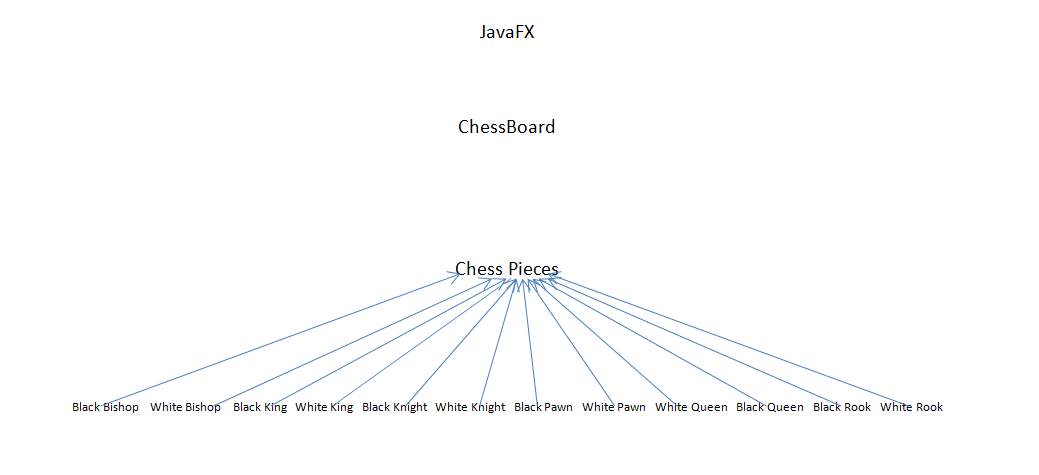
**1. Program Concept**

The objective of chess is to get the opponent’s king piece into a position such that the any next  move the opponent makes will result in the king piece’s capture. If the king cannot make any legal moves (i.e. all potential spots are already occupied the game is tied). Each piece moves in a distinct way with some pieces moving differently to take pieces. If the opposing king is in a situation where he can be taken, check is called and the other player must make a move that does not place the king in check.

|  |  |
| --- | --- |
| Functionality Name | Description |
| Moving Pieces | The user can move a piece on the board |
| 2 player | 2 users can play the game using the same board |
| Game buttons | Start game/ quit buttons are provided |
| Promotion | Allows the user to select what piece the user can promote a pawn to |
| Pieces taken | The console will output the pieces already taken on the GUI |

**2. Code structure**

**2.1 Class diagram**



**2.2 Briefly describe the overall structure of your code**

The javaFx class that we made runs the code to update the chessboard (moving pieces and checking for check, checkmate and stalemate). It also updates the large canvas which we used to display the board and the two smaller canvases that display the pieces that have been taken by both sides. The javaFx class calls the turn function inside of the abstract class ChessPieces which uses the posValid method inside of ChessBoard to help update the board.

Many of the functions called passing the individual types of pieces (one of each for black and white) used the abstract parent class of ChessPieces and the inherited/overwritten method isWhite.

**2.3 What are the data structures you used? Why did you used that specific data structure?**

Generic ArrayLists were used as the data structures for the program.

This data structure was selected because it provided us with capabilities of keeping track of the White/Black pieces, the already taken White/Black pieces. It also allowed us to keep the code simpler when we wanted to find, remove, or replace(in the case of a pawn’s promotion) a certain piece.

**2.4 What is the major algorithms of your program?**

Check:

* Check will see whether opponent team’s king can be attacked by our team’s pieces.
* This will be accomplished by checking the valid moves our team’s pieces can make in relation to opponent team’s king.

causeCheck(int xPos, int yPos, ArrayList<ChessPieces> opponent)

if(xPos || yPos out of bounds)

return false

else

   check if any opposing piece stored in opponent can take the king with their next move

if true return true

return false

Turn:

* This function will check whether or not you can move a particular piece. If yes, then it will move it or if not then it will not move the piece and output an invalid message.

turn(ChessPieces toMove, int xPos, int yPos)

         if(causeCheck for the location for the current king)

if(move does not take the king out of check)

                 invalid move

         else

check if move is valid for the piece to be moved

               if true then move is valid

otherwise invalid move

      if(a pawn reached till the end of the opponent’s side)

          promote the pawn to the desired piece

     check to see if move puts opponent in check, checkmate or stalemate

update whose move it is

**3. Test**

**3.1 Unit tests**

|  |  |
| --- | --- |
| **Test Case Name** | **Purpose** |
| Piece Selection | Tests if the canvas properly select the desired piece  Inputs: Mouse click event  Outputs: Error Message if no piece there or selects piece if piece belongs on current user’s team |
| Movement | Tests to see if the pieces move in the correct way according to the rules of Chess.  Inputs: Mouse clicks on piece and desired location  Outputs: Either move the piece or print out an error message |
| Capturing | Tests to see if pieces can be taken and are correctly placed in the proper taken “pile” on the GUI  Inputs: Piece to be taken and piece moving to that location  Outputs: Piece taken is placed into the correct taken pile and the piece moving there is moved to that location in the grid |

**3.2 System tests**

|  |  |
| --- | --- |
| **Test Case Name** | **Purpose** |
| Check | Checks if opposing king has been placed into check  Inputs: Opposing king’s position on the board  Outputs: Check is printed if the king has been placed into a capture spot for an opposing piece, false otherwise |
| Checkmate | If opposing king is in a position where he is placed in check by a recent move, it checks whether or not the king will be taken no matter what move he makes next  Inputs: Opposing king’s position and surrounding spots  Outputs: Checkmate is printed if no moves that do not put the king in check can be made, the game ends and the start button is re-enabled, otherwise nothing is printed |
| Stalemate | If opposing king is in a position where he is placed in check by a recent move, it checks whether or not the king can make any move.  Inputs: Opposing king’s position and surrounding spots  Outputs: Stalemate is printed in there are no moves the king can make,the game ends and the start button is re-enabled, otherwise nothing is printed. |